RC1000-PP

Software User Guide

Xilinx, XBLOX and XACTStep are trademarks of Xilinx Corp.
Microsoft and MS-DOS are registered trademarks and Windows, Windows 95 and Windows NT are trademarks of Microsoft Corporation.

This manual was written by Matthew Bowen.

© Celoxica Limited. All rights reserved
Version 1.20

Table of Contents

Conventions v

1. INTRODUCTION 1-1

1.1 About This Manual 1-2

1.2 About the Software 1-3

2. INSTALLATION 2-1

2.1 System Requirements 2-2

2.2 Installing the RC1000-PP Support Software 2-3

2.3 Installing the Documentation Browser 2-6

2.4 Directory Structure 2-7

3. EXAMPLE PROGRAMS 3-1

3.1 Introduction 3-2

3.2 Checking the RC1000-PP Hardware 3-3

3.3 The addone Example Program 3-5

3.4 The dma Example Program 3-7

3.5 The video Example Program 3-9

3.6 Compiling the Example Programs 3-10

4. USING THE HOST SUPPORT SOFTWARE 4-1

4.1 Introduction 4-2

4.2 Basic Concepts 4-3

4.3 Using the Host Interface Library 4-5

4.4 Initialising the hardware and software 4-6

4.5 Communicating with the FPGA 4-9

4.6 Cleaning up the Board 4-12

5. USING THE HANDEL-C SUPPORT SOFTWARE 5-1

5.1 Introduction 5-2

5.2 Writing FPGA Programs in Handel-C 5-3

5.3 Communicating with the Host 5-5

6. UTILITIES 6-1

6.1 Introduction 6-2

6.2 The list Utility 6-3

6.3 The setid Utility 6-4

6.4 The gencfg Utility 6-5

6.5 The loadfpga Utility 6-7

6.6 The diag Utility 6-8

Conventions

A number of conventions are used in this document. These conventions are detailed below.

Warning Message. These messages warn you that actions may damage your hardware.

Handy Note. These messages draw your attention to crucial pieces of information.

Hexadecimal numbers appear in this document. They are prefixed with ‘0x’ (in common with standard C syntax).

Sections of code or commands that you must type are given in typewriter font like this:

```
void main();
```

Information about a type of object you must specify is given in italics like this:

copy *SourceFileName DestinationFileName*

1. Introduction

1.1 About This Manual

This manual provides details of the installation and use of the support software for the Celoxica's RC1000-PP reconfigurable computer platform. Details of the support functions are given in the RC1000-PP Function Reference Manual and details of the hardware are given in the RC1000-PP Hardware Reference Manual and extensive references are made throughout to these companion documents.

This chapter gives an overview of the RC1000-PP system.

Chapter 2 details how to install the software support on your PC. For details of installing the hardware, consult the RC1000-PP Hardware Reference Manual.

Chapter 3 details the example programs supplied with the RC1000-PP package. These examples are designed to show how to write host programs that talk to the RC1000-PP hardware and how to write FPGA programs that talk to the host PC. The FPGA example programs are written using the Handel-C language.

Chapter 4 describes the support offered by the RC1000-PP software package in more detail. This support includes host PC libraries and FPGA macros.

Chapter 5 describes the host utilities supplied with the RC1000-PP package. These utilities provide easy handling of FPGA configuration files and enable the RC1000-PP FPGA to be configured without writing a host support program.

1.2 About the Software

The RC1000-PP Support Software provides host libraries to simplify the process of initialising and talking to the hardware. A number of example programs and utilities are also provided to serve as a starting point for the development of your own applications.

The software provides a number of groups of host functions:

- Initialisation functions
- Functions to handle FPGA configuration files
- Functions to control the RC1000-PP programmable clock
- Functions to transfer data to and from the RC1000-PP FPGA
- Functions to help with error checking and debugging

The host software comes in the form of a static C library which can be linked to host programs and a set of Handel-C macros for use with FPGA programs.

2. Installation

2.1 System Requirements

The RC1000-PP support software requires the following platform:

- IBM PC compatible computer.
- Windows 98 or NT 4.0 operating system.
- Microsoft Visual C++ version 4.0 or later.
- 16Mb RAM.
- Up to 10Mb hard disk space.
- CD ROM drive.

You will also need tools to target the FPGA. Celoxica Ltd recommend using the Handel-C compiler and Xilinx Alliance or Foundation software. Contact Celoxica Ltd for further details of these products.

2.2 Installing the RC1000-PP Support Software

This section details how to install the RC1000-PP support software.

For dual boot systems, simply follow both the Windows 98 and NT4.0 instructions below, specifying the same installation directory. Host applications can be written to execute on either Windows 98 or Windows NT since the software interface is identical across the two platforms.

To install for Windows 98:

1. Install the hardware as described in the RC1000-PP Hardware Reference Manual. Start Windows 98.

2. Windows 98 should detect that new hardware has been added to the system. When asked whether you would like to have Windows find a driver or whether you would like to select from a list choose to select from a list of drivers.

3. Select 'Other devices' from the first list.

4. Select 'Have Disk' when given a list of devices.

5. Insert the RC1000-PP Support Software CD in the CD-ROM drive and browse for the CD ROM. The driver information file, **`rc1000pp.inf`** is in the **`win98`** directory of the CD ROM.

6. Select the RC1000-PP FPGA Co-processor driver from the list. The **`rc1000pp.sys`** file is in the **`win98`** directory of the CD ROM. Windows should then continue to start.

7. When Windows has started, open an MS-DOS window.

8. From the DOS prompt, run the **`setup`** program in the **`win98`** directory of the CD. E.g., if your CD ROM drive is drive **`d:`** type:

   `d:\win98\setup`

9. Follow the on screen instructions.

10. It is recommended (but not necessary) that the utilities directory be added to your system path. You can do this by editing your **autoexec.bat** file to include the following line:

```
PATH="c:\Program Files\Celoxica
Ltd\RC1000-PP\utils";%PATH%
```

(This should all appear on a single line in the **autoexec.bat** file. You should modify this line to match the installation directory you chose during setup.)

To install for Windows NT 4.0:

1. Install the hardware as described in the RC1000-PP Hardware Reference Manual. Start Windows NT.

2. Insert the RC1000-PP Support Software CD in the CD-ROM drive.

3. Open an MS-DOS window.

4. From the DOS prompt, run the **`setup`** program in the **`winnt`** directory of the CD. E.g., if your CD ROM drive is drive **`d:`** type:

   `d:\winnt\setup`

5. Follow the on screen instructions.

6. It is recommended (but not necessary) that the utilities directory be added to your system path. You can do this from the Start ➨ Settings ➨ ControlPanel ➨ System ➨ Environment window. The following should be added to the PATH environment variable:

   `c:\Program Files\Celoxica Ltd\RC1000-PP\utils`

   (You should modify this line to match the installation directory you chose during setup.)

By default, the support software is installed in the directory **`\Program Files\Celoxica Ltd\RC1000-PP`**. All directory paths given in this manual start from the **`RC1000-PP`** directory.

2.3 Installing the Documentation Browser

The Handel-C documentation is provided in a format suitable for the Adobe Acrobat Reader software. If you do not already have this software installed, you should follow these instructions to install it.

1. Insert the RC1000-PP Support Software CD in your CD-ROM drive.

2. Open an MS-DOS window.

3. From the prompt run the **ar32e30** program in the root directory of the CD. For example, if your CD ROM is drive **d:** then type:

   d:\ar32e30

4. Follow the on screen instructions

For the latest version download from the http://www.adobe.com web site.

2.3.1 Accessing the documentation

The documentation is in three files:

SWref.pdf	Software User Guide (this document).
FNref.pdf	Software Function Reference Manual.
HWref.pdf	Hardware Reference Manual

If you chose to install the documentation on your hard disk when you installed the RC1000-PP Support Software, the documentation files will be stored in the **doc** subdirectory.

The documentation files are also stored in the root directory of the RC1000-PP Support Software CD-ROM.

2.4 Directory Structure

Upon successful installation, the following directories and files should have been created on your hard disk. If you chose not to perform a typical installation, some of these files may be missing.

README.TXT	Text file containing latest information.
examples##	Multiple examples directories for each type of FPGA.
➥ **addone.exe**	Simple example to add one to a number.
➥ **dma.exe**	Simple example to DMA data to and from a card.
➥ **examples.dsw**	Microsoft Visual C++ 6.0 workspace file for example programs.
➥ **fpga**	Subdirectory containing FPGA files for example programs.
➥ **video.exe**	More advanced example program to demonstrate real time video processing.
doc	
➥ **FNref.pdf**	RC1000-PP Function Reference Manual.
➥ **HWref.pdf**	RC1000-PP Hardware Reference Manual.
➥ **SWref.pdf**	RC1000-PP Software User Guide (this document).
fpga	
➥ **v100**	Handel-C directory for revision 1.0 PCBs.
➥ **XC4000**	Handel-C directory for XC4000 FPGAs.
➥ **Virtex**	Handel-C directory for Virtex FPGAs.
include	
➥ **pp1000.h**	Conventional C header file for RC1000-PP host support software
lib	
➥ **pp1000.lib**	RC1000-PP support software host library.
utils	
➥ **gencfg.exe**	FPGA configuration to C header file conversion utility.
➥ **diag.exe**	Host diagnostics utility.
➥ **list.exe**	Host utility to list the cards in a system.
➥ **loadfpga.exe**	Host RC1000-PP configuration utility.
➥ **setid.exe**	Host utility to set a card ID.
➥ **edifmake.bat**	Batch file to build FPGA images from EDIF files.

3. Example Programs

3.1 Introduction

This chapter details the example programs provided with the RC1000-PP Support Software Package. The purpose of the example programs is twofold. Firstly, they provide a quick means of testing that the RC1000-PP hardware is installed and functioning correctly. Secondly, they provide a starting point for your own programs.

Three example programs are provided. These are:

- A simple program that prompts for input of a number and uses the FPGA to calculate the value of the number plus one.
- A simple program that demonstrates the use of the DMA functions to transfer data to and from the RC1000-PP SRAM.
- A more advanced program that demonstrates using the FPGA to perform real-time video processing.

3.2 Checking the RC1000-PP Hardware

After installation of your RC1000-PP hardware following the instructions in the RC1000-PP Hardware Reference Manual and installation of the support software as described in the previous chapter, it is recommended that you run a simple test program provided to ensure that your hardware is functioning correctly.

The test program simply configures the FPGA and flashes the LEDs on the board. To run the test, open a DOS prompt, change to the **`utils`** subdirectory and type the following:

```
loadfpga –i CardID –c 20000000 flash##.bit
```

Here, you should replace *CardID* with the ID of the board. The IDs of the cards in the system can be determined by typing the following:

```
list
```

This should result in the following (or similar) output:

```
Number of cards in system : 1

Card ID : 2        Serial Number : 0x00000003
```

You should also replace the ## in the bit filename with a number relating to the FPGA fitted to your board. Currently supplied files are:

PCB revision	FPGA fitted	Filename
1.0	4085XL	**`flash85-1.bit`**
2.0	4085XL	**`flash85.bit`**
2.0	40150XV	**`flash150.bit`**
2.0	40200XV	**`flash200.bit`**
2.0	40250XV	**`flash250.bit`**
2.0	Virtex V400	**`flashv400.bit`**
2.0	Virtex V600	**`flashv600.bit`**
2.0	Virtex V800	**`flashv800.bit`**
2.0	Virtex V1000	**`flashv1000.bit`**
4.0	Virtex V1000E	**`flashv1000e.bit`**
4.0	Virtex V1600E	**`flashv1600e.bit`**
4.0	Virtex V2000E	**`flashv2000e.bit`**
4.0	Virtex V405E	**`flashv405e.bit`**
4.0	Virtex V812E	**`flashv812e.bit`**

The PCB revision is printed on the RC1000-PP card.

In this example, the card ID of the only board in the system is 2 and, assuming that a 40150XV part is fitted to the board, you should type:

```
loadfpga –i 2 –c 20000000 flash150.bit
```

This test can also be used to check the programmable clock on the board. By changing the value of the clock frequency in Hz (with the **–c** option), you should see the rate of the flashing increase or decrease. For example, to halve the rate of flashing, type:

```
loadfpga –i 2 –c 10000000 flash150.bit
```

Should you be unable to get this test program to run, you should check the installation of both the hardware and software. If it still does not work correctly, contact Celoxica Ltd by email at:

support@celoxica.com

3.3 The addone Example Program

The **addone** example program uses the FPGA to add one to numbers typed in by the user. This example illustrates setting up the board and software, configuring the FPGA from a file, setting the programmable clock and using bi-directional single byte communications.

There is a separate copy of each of the examples for each FPGA type that can be fitted to a board. To run the examples, you should open a DOS box and move to one of the following directories:

PCB revision	FPGA fitted	Directory
1.0	4085XL	**examples85-1**
2.0	4085XL	**examples85**
2.0	40150XV	**examples150**
2.0	40200XV	**examples200**
2.0	40250XV	**examples250**
2.0	Virtex V400	**examplesv400**
2.0	Virtex V600	**examplesv600**
2.0	Virtex V800	**examplesv800**
2.0	Virtex V1000	**examplesv1000**
4.0	Virtex V1000E	**examplesv1000e**
4.0	Virtex V1600E	**examplesv1600e**
4.0	Virtex V2000E	**examplesv2000e**
4.0	Virtex V405E	**examplesv405e**
4.0	Virtex V812E	**examplesv812e**

The PCB revision is printed on the RC1000-PP card.

To run the program, type the following:

addone

The program will repeatedly ask you for a number and then return the number plus one back to you. To stop the program, enter zero as the number.

The **addone.c** host program performs the following actions:

1. Open and initialise the RC1000-PP card.
2. Set the programmable clock to run at 20MHz.
3. Configure the FPGA from the **addone.bit** file.
4. Repeatedly write numbers to and read results back from the FPGA.

The **addone.c** Handel-C program performs the following actions:

1. Repeatedly read numbers from and return results to the host program.

3.4 The dma Example Program

The **dma** example program demonstrates the use of the DMA functions to transfer data to and from the SRAM banks on the RC1000-PP. This example illustrates setting up the board and software, requesting memory banks and DMAing data to and from the memory banks.

There is a separate copy of each of the examples for each FPGA type that can be fitted to a board. To run the examples, you should open a DOS box and move to one of the following directories:

PCB revision	FPGA fitted	Directory
1.0	4085XL	**examples85-1**
2.0	4085XL	**examples85**
2.0	40150XV	**examples150**
2.0	40200XV	**examples200**
2.0	40250XV	**examples250**
2.0	Virtex V400	**examplesv400**
2.0	Virtex V600	**examplesv600**
2.0	Virtex V800	**examplesv800**
2.0	Virtex V1000	**examplesv1000**
4.0	Virtex V1000E	**examplesv1000e**
4.0	Virtex V1600E	**examplesv1600e**
4.0	Virtex V2000E	**examplesv2000e**
4.0	Virtex V405E	**examplesv405e**
4.0	Virtex V812E	**examplesv812e**

The PCB revision is printed on the RC1000-PP card.

The **dma** example program assumes that there is only one RC1000-PP board fitted in the system.

To run the program, type the following:

```
dma
```

The **dma.c** host program performs the following actions:

1. Open and initialise the RC1000-PP card.
2. Request bank 0 on the card.
3. DMA data to and from the card and check returned data is correct.
4. Report the average data transfer rate to and from the card.

The data transfer rates reported are highly machine dependent but figures of greater than 100 Mb/s should be reported for modern PCs.

3.5 The video Example Program

The **video** example program uses the FPGA to perform a warping video processing operation. This example illustrates setting up the board and software, configuring the FPGA from a linked-in FPGA image array, setting the programmable clock and bi-directional communications using both single byte and DMA transfers. It also illustrates the power of the FPGA as a processing unit when coupled with the Handel-C programming language.

There is a separate copy of each of the examples for each FPGA type that can be fitted to a board. To run the examples, you should open a DOS box and move to one of the following directories:

PCB revision	FPGA fitted	Directory
1.0	4085XL	**examples85-1**
2.0	4085XL	**examples85**
2.0	40150XV	**examples150**
2.0	40200XV	**examples200**
2.0	40250XV	**examples250**
2.0	Virtex V400	**examplesv400**
2.0	Virtex V600	**examplesv600**
2.0	Virtex V800	**examplesv800**
2.0	Virtex V1000	**examplesv1000**
4.0	Virtex V1000E	**examplesv1000e**
4.0	Virtex V1600E	**examplesv1600e**
4.0	Virtex V2000E	**examplesv2000e**
4.0	Virtex V405E	**examplesv405e**
4.0	Virtex V812E	**examplesv812e**

The PCB revision is printed on the RC1000-PP card.

To run the program, type the following:

```
video
```

You should see a dynamically warped picture in the resulting window. After a few frames, an average frame rate is reported.

The video example program will run best in a 15 bit per pixel video mode (32768 colours). While it will run correctly at other colour depths, the colour conversion will reduce the frame rate.

3.6 Compiling the Example Programs

This section details the re-compilation process for the example programs.

The **addone** and **video** examples can be split into two parts – the host program and the FPGA program. The **dma** example is a stand-alone host program. The RC1000-PP support software package provides pre-compiled versions of the example programs. This section describes how to re-compile the example programs should you modify them in any way.

3.6.1 Compiling the Host Example Programs

The host programs are all written using the Microsoft Visual C++ compiler version 6.0. Workspace files are provided for this compiler so if you are using an earlier version of the compiler you will need to create new project files to contain the source files provided.

The **examples.dsw** file can be used to re-compile the examples.

3.6.2 Compiling the FPGA Example Programs

The FPGA programs are all written for the Handel-C language. A compiler for this language is available from Celoxica Ltd. Contact Celoxica Ltd for further details of the Handel-C language and compiler.

The RC1000-PP support software requires the use of version 2.1 or later of the Handel-C compiler.

The FPGA programs can all be re-compiled with the command line:

handelc –edif *Program* **–cpp -I***IncludeDir*

where *Program* is the name of the Handel-C program to compile and *IncludeDir* is the **fpga** directory containing the Handel-C header file for the RC1000-PP support software.

The Handel-C compiler will generate a Xilinx format netlist file with the suffix **.edn** which must then be passed through the Xilinx place and route tools. A batch file is provided in the **utils** directory to help with this stage of the process. To use it, you should type:

edifmake *Program*

In both cases, the program name should be specified without extensions.

3.6.3 Converting the FPGA Configuration Files

The Xilinx place and route tools will generate FPGA configuration files with a **.bit** suffix. As will be described in chapter 4, the RC1000-PP host support software provides functionality enabling configuration files to be **#include**d in host programs. To achieve this, the binary configuration file must be converted to a C header file which can be **#include**d in the host program. The **video** example program uses this approach and so its **warp.bit** configuration file must be converted in this way. (The **addone.exe** example program loads the **addone.bit** file directly.)

The RC1000-PP support software package includes a utility to convert FPGA configuration files to C header files called **gencfg.exe**. Chapter 6 describes this utility in greater detail but its basic syntax is:

gencfg *SourceFile DestinationFile*

Here, *SourceFile* is the **.bit** file generated by the Xilinx place and route tools and *DestinationFile* is the filename of the C header file to generate. There is a file called **warp.h** in the **fpga** subdirectory which is the output from the **gencfg** utility. Should you re-compile the **warp.c** Handel-C program, you must convert the resulting bit file and re-compile the **video.exe** host program.

To perform these actions, you must follow these stages:

```
handelc –edif warp.c –cpp –I..\..\fpga
edifmake warp
gencfg warp.bit warp.h
```

You must then use Microsoft Visual C++ to re-compile the **video.exe** host program.

4. Using the Host Support Software

4.1 Introduction

This chapter details the components of the RC1000-PP host support software library and describes step-by-step how to write your own programs for use with the RC1000-PP board.

Chapter 5 describes the Handel-C support software and how to write Handel-C programs for the FPGA.

4.2 Basic Concepts

This section details the basic knowledge required to write a host program for the RC1000-PP board.

4.2.1 Identifying an RC1000-PP Card

Each RC1000-PP card has two means of identification. Firstly, the serial number which is factory programmed into the card and is guaranteed to be unique across all the cards produced.

Secondly, a card ID is programmed into the card. This ID can be set by the user with the **`setid`** utility and is the preferred means of identifying a card.

To illustrate the use of the card ID, consider a system with two RC1000-PP cards. Card A has an input daughter module fitted and card B has an output daughter module fitted. The problem is how to differentiate between the two boards? One solution would be to obtain the serial numbers of each board and identify the boards with these numbers. However, suppose a second, identical system is set up with two new boards. These new boards will have different serial numbers and the host program would have to be modified to use these new serial numbers.

Now suppose that card A has an ID of 1 programmed by the user and card B has an ID of 2. These IDs can be used to identify the cards in the host program. The second system can also have its input card ID set to 1 and its output card ID set to 2 by the user so the host program need not change. The user simply needs to program the card IDs on each system that is set up.

The host support software uses the card ID to identify a card when creating a handle for it. It also provides a function for querying which cards are in the system.

4.2.2 Configuring the FPGA

The host support software provides a set of functions to configure the FPGA on an RC1000-PP board. There are three ways of configuring from FPGA image files:

1. Configuring directly from a file.
2. Loading an FPGA image file into memory ready for configuration later.
3. Including a static array containing the configuration information in the host code ready for configuration later.

The first method requires only a single function call but must re-load the configuration file from disk every time the FPGA is configured.

The second method allows the configuration file to be loaded once and used many times.

The third method removes the need for separate configuration files and allows the configuration image to be linked in with the host executable.

4.2.3 Communicating with the FPGA

There are three methods of communicating with the FPGA.

1. Single bit signalling using 2 pins on the FPGA.
2. Single byte data transfers using the control/status ports on the RC1000-PP.
3. Bulk data transfers using the DMA controller and the banks of SRAM.

The first method can be used to signal a state to the FPGA or to the host.

The second method can be used to send short control messages to the FPGA or short status messages from the FPGA.

The third method is recommended for large data transfers. The RC1000-PP card has up to 4 banks of SRAM fitted. Each bank can be granted to either the host or the FPGA (but not both) at any one time. When a memory bank is granted to the host, the DMA controller can transfer data between the host memory and the SRAM memory bank. When a memory bank is granted to the FPGA, it can access the data in the memory to read source data from the host or fill in return data to the host. One of the first two methods of communication can be used to synchronise the swapping of ownership of a memory bank.

For example, you can use the following sequence to transfer a block of data from the host to the FPGA:

Host Action	FPGA Action
1. Build data in host memory buffer	1. Release ownership of RC1000-PP memory bank
2. Request ownership of R1000-PP memory bank	2. Wait for control word from host
3. Transfer data to RC1000-PP memory bank	
4. Release ownership of RC1000-PP memory bank	
5. Send control word to FPGA	
	3. Request ownership of RC1000-PP memory bank
	4. Process data from host

4.3 Using the Host Interface Library

The RC1000-PP support software is provided in the form of a statically linked library. This consists of two files:

File	Description
pp1000.h	C header file
pp1000.lib	Support library

Any host program that uses the RC1000-PP support software must include the **pp1000.h** header file by adding the following line to the start of the program:

```
#include "pp1000.h"
```

This assumes that the **include** directory of the RC1000-PP support software is in the C compiler's include search path. The include search path in Microsoft Visual C++ V6.0 can be set from the Tools ➡ Options ➡ Directories menu. For other compilers, refer to your compiler documentation for details of how to set the include search path.

The host program must also be linked with the **pp1000.lib** library file. To link the library file with your host program, you must add the **.lib** file to your linker command line. To add the file using Microsoft Visual C++ V6.0, type the name of the file in the Object/library modules edit box in the Project ➡ Settings ➡ Link ➡ General dialog box. For other compilers, refer to your compiler documentation for details of how to link additional libraries.

4.4 Initialising the hardware and software

The first step for any program is to initialise the RC1000-PP hardware and support software. This is done by calling the **PP1000OpenCard()** function specifying the card's ID. This function will return a handle which must be used to identify the RC1000-PP board in future calls to the support software.

If the exact card that is opened is not important, call the **PP1000OpenFirstCard()** to obtain a handle to the first free card in the system.

The **PP1000GetCards()** function can be used to find out the ID of all the cards in the system. The following code fragment illustrates its use:

```
unsigned long NumCards;
unsigned long Count;
PP1000_DEVICE_COUNT *Present;

PP1000GetCards(&NumCards, &Present);
for (Count=0; NumCards>0; Count++)
{
    printf("Card ID : %d, number present : %d\n",
                          Present[Count].CardID,
                          Present[Count].Count);
    NumCards -= Present[Count].Count;
}
free(Present);
```

Note that it is strongly recommended that you check all return codes from library functions. These checks are omitted here for clarity.

4.4.1 Setting the programmable clock rate

The RC1000-PP board has two programmable clocks as two of the clock sources for the FPGA (the others being a fixed PCI bus clock and a clock input connector – see the Hardware Reference and Function Reference Manuals for how to select the clock). The RC1000-PP support software provides a function to set the programmable clock period for FPGA designs that use this clock. This rate should be set before configuring the FPGA to ensure that the design is not over-clocked when it first starts.

To set the programmable clock rate, use the **PP1000SetClockRate()** function specifying the handle of the board and the required rate in Hertz. For example, to set the clock to 10MHz you could use the following code:

```
Status=PP1000SetClockRate(Handle,
                          PP1000_MCLK, 10e6);
if (Status!=PP1000_SUCCESS)
{
    ReportError(Status);
}
```

The **PP1000_MCLK** parameter tells the support software to program the MCLK clock generator. Refer to the Hardware Reference and Function Reference Manuals for details of the other clock sources on the board.

4.4.2 Configuring the FPGA

Once the board has been initialised and the clock set up, you must configure the FPGA with your FPGA image file. There are three ways of configuring from FPGA image files:

1. Configuring directly from a file using the **PP1000ConfigureFromFile()** function.

2. Loading an FPGA image file into memory with the **PP1000LoadFile()** function ready for configuration later with the **PP1000ConfigureFPGA()** function.

3. Including a static array containing the configuration information in the host code. This can be registered with the support software with the **PP1000RegisterImage()** function to obtain a handle for use later with the **PP1000ConfigureFPGA()** function.

The first option is the simplest but requires re-reading the configuration file every time the FPGA is configured which can be a significant overhead when frequently re-configuring the FPGA. The second option overcomes this limitation while the third option allows fast configuration without the need for multiple files. The third option means that a single executable can be used which contains the host program and the FPGA configuration information.

The RC1000-PP support software package contains a utility called **gencfg** to generate static arrays from FPGA configuration files to help with option 3 above. Refer to chapter 6 for further details of this utility.

All file handling routines read configuration files in Xilinx binary BIT format (**.bit** extension)

For example, to configure the FPGA directly from a file called **config.bit** you could use the following code:

```
Status=PP1000ConfigureFromFile(Handle,
                               "config.bit");
if (Status!=PP1000_SUCCESS)
{
    ReportError(Status);
}
```

To read the file into memory and then configure later you could use the following code:

```
PP1000_IMAGE Image;

Status=PP1000LoadImage("config.bit", &Image);
if (Status!=PP1000_SUCCESS)
{
    ReportError(Status);
}

......

Status=PP1000ConfigureFPGA(Handle, Image);
if (Status!=PP1000_SUCCESS)
{
    ReportError(Status);
}
```

To register a static image array and then configure later you could use the following code:

```
PP1000_IMAGE Image;

Status=PP1000RegisterImage(configBuffer,
                           configLength,
                           &Image);
if (Status!=PP1000_SUCCESS)
{
    ReportError(Status);
}

......

Status=PP1000ConfigureFPGA(Handle, Image);
if (Status!=PP1000_SUCCESS)
{
    ReportError(Status);
}
```

4.5 Communicating with the FPGA

Once the FPGA has been configured, there are three ways to communicate with it.

4.5.1 Single Bit Communications with the FPGA

The RC1000-PP support software provides two functions to control single pins on the FPGA. The **PP1000SetGPO()** function sets the state of a pin on the FPGA and the **PP1000ReadGPI()** function reads the state of a second FPGA pin. The following code fragment illustrates how to set the state of the GPO pin to 1.

```
Status=PP1000SetGPO(Handle, 1);
if (Status!=PP1000_SUCCESS)
{
    ReportError(Status);
}
```

The following code fragment illustrates how to read the state of the GPI pin.

```
Status=PP1000ReadGPI(Handle, &Value);
if (Status!=PP1000_SUCCESS)
{
    ReportError(Status);
}
```

Refer to section 5.3.1 for details of how to set and read the states of these pins from a Handel-C program on the FPGA.

The GPO pin is equivalent to the USERO pin and the GPI pin is equivalent to the USERI pin.

4.5.2 Single Byte Communications with the FPGA

The RC1000-PP board has a single byte wide port in either direction between the host and FPGA. This port can be used to send short messages as control or status bytes between the two parties. The **PP1000WriteControl()** function will send a byte from the host to the FPGA and the **PP1000ReadStatus()** function will wait for a byte to be sent by the FPGA.

Both functions are blocking and will only return when the operation has completed.

Refer to section 5.3.2 for details of how to send and receive single bytes in a Handel-C program on the FPGA.

Refer to the **addone** example program for code samples.

4.5.3 DMA Data Transfers to and from the RC1000-PP SRAM

Bulk data transfers between the host and the RC1000-PP SRAM can be performed by using the DMA support functions in the support software. A DMA transfer consists of a number of stages:

1. Set up a DMA channel on a user buffer.
2. Request access to the required memory bank(s).
3. Do the DMA transfer.
4. Release the memory bank(s).
5. Free the DMA channel.

The transfer can be either a contiguous block of memory (1D transfer) or a series of short transfers with gaps in between (2D transfer). The 2D transfer is intended for transferring image data with an image width, height and pitch.

To set up a 1D transfer, call the **PP1000SetupDMAChannel()** function. To set up a 2D transfer, call the **PP1000Setup2DDMAChannel()**.

The memory banks required for the data transfer can be requested with the **PP1000RequestMemoryBank()** function. This function will wait until the memory banks have been granted to the host before returning. You should ensure that the FPGA has released its request for the requested memory banks so that the host can gain access to them. Refer to section 5.3.3 for details of requesting and releasing memory banks in a Handel-C FPGA program.

The **PP1000DoDMA()** function should be called to start the DMA transfer. This function will only return when the DMA transfer is complete.

The **PP1000ReleaseMemoryBank()** function can be called to release the ownership of the memory banks. Once ownership has been released, the FPGA can gain access to the memory banks. Refer to section 5.3.3 for details of requesting and releasing memory banks in a Handel-C FPGA program.

The **PP1000CloseDMAChannel()** function should be called to free up the resources associated with a DMA channel handle.

For example, the following code fragment could be used to transfer 1Mb of contiguous data from **`Buffer`** to address 0x1000 in bank 1

of an RC1000-PP. Note that the card address is 0x201000 since bank 0 is 2 Mb long.

```
// Stage 1: set up DMA channel
PP1000SetupDMAChannel(Handle,  // card handle
                      Buffer,  // host address
                      0x201000,// card address
                      0x100000,// length
                      PP1000_PCI2LOCAL,
                              // write to card
                      &Channel);
                              // return handle

// Stage 2: request bank 1 (mask = 0b0010=0x2)
PP1000RequestMemoryBank(Handle, 0x2);

// Stage 3: do the DMA
PP1000DoDMA(Channel);

// Stage 4: release bank 1
PP1000ReleaseMemoryBank(Handle, 0x2);

// Stage 5: free DMA channel resources
PP1000CloseDMAChannel(Channel);
```

If multiple transfers on the same DMA channel are required, the channel may be left open (i.e. don't call **PP1000CloseDMAChannel()**). However, leaving channels open may have an adverse effect on system performance as less memory will be available for virtual memory paging. Therefore, DMA channels should be closed as soon as possible after a DMA transfer.

Note that it is strongly recommended that you check all return codes from library functions. These checks are omitted here for clarity.

Refer to the **dma** example program for more code examples of DMA transfers.

4.6 Cleaning up the Board

Before exiting from your host program, you should free any FPGA images you allocated with the **PP1000LoadFile()** or **PP1000RegisterImage()** functions by calling the **PP1000FreeImage()** function for each one. For example, to free the image **Image**, you could use the following code:

```
Status=PP1000FreeImage(Image);
if (Status!=PP1000_SUCCESS))
{
    ReportError(Status);
}
```

You should then close the board with the following code:

```
Status=PP1000CloseCard(Handle);
if (Status!=PP1000_SUCCESS))
{
    ReportError(Status);
}
```

5. Using the Handel-C Support Software

5.1 Introduction

This chapter details the components of the RC1000-PP Handel-C support software library and describes step-by-step how to write your own programs for use with the RC1000-PP board.

5.2 Writing FPGA Programs in Handel-C

The RC1000-PP support software package contains a Handel-C include file containing a number of macros to allow simple communications to and from the host. There are three versions of the Handel-C support routines. There are all called **pp1000.h** but are stored in different directories as follows:

Include file	PCB Revision	FPGA types
fpga\v100\pp1000.h	V1.0	4085XL
fpga\xc4000\pp1000.h	V2.0	4085XL, 40150XV, 40200XV, 40250XV
fpga\virtex\pp1000.h	V2.0 or later	Virtex V400, V600, V800 V1000, V1000E, V1600E, V2000E, V405E, and V812E

The RC1000-PP support software requires the use of version 2.1 or later of the Handel-C compiler.

The Handel-C support software can be customised by using one or more of the following lines before including the **pp1000.h** header file. First, the clock division factor must be set up for the asynchronous RAM access macros. One of the following lines must appear at the start of the program.

Source Code	Effect
#define PP1000_DIVIDE1	Sets clock division of 1. RAM WE pulse is placed in second half of the clock cycle.
#define PP1000_DIVIDE3	Sets clock division of 3. RAM WE pulse is placed in middle of the Handel-C clock cycle.
#define PP1000_DIVIDE4	Sets clock division of 4. RAM WE pulse is placed from ½ to ¾ of the way through the Handel-C clock cycle.

Refer to the Handel-C Language Reference Manual for details of accessing asynchronous SRAM.

The following line will set the external clock source to be VCLK instead of MCLK. Refer to the Hardware Reference and Function Reference Manuals for details of clock sources.

```
#define PP1000_CLOCK PP1000_VCLK
```

The RAM access macros can be set to access the external SRAM either as 8 bit wide or 32 bit wide memory. One of the following lines must appear at the start of the Handel-C program.

Source Code	Effect
#define PP1000_8BIT_RAMS	Access external SRAM as 8 bits x 2-8Mbytes.
#define PP1000_32BIT_RAMS	Access external SRAM as 32 bits x 512Kwords-2Mwords.

In addition, the FPGA family and part type must be set in the Handel-C program to match the FPGA fitted to the target board.

For example, the following code fragment could be used at the start of a Handel-C program:

```
#define PP1000_32BIT_RAMS
#define PP1000_DIVIDE4
set family = Xilinx4000XV;
set part = “40150XVBG560-09”;
#include “pp1000.h”
```

You must also inform the Handel-C compiler of the location of the header file. This can be done either by adding the option:

-cpp –I*RC1000PPDir***fpga***SubDirectory*

to the Handel-C command line or by adding the same text to the **HANDELC_CPPFLAGS** environment variable. Refer to the Handel-C compiler documentation for further details of the compiler command line options.

Your should replace *SubDirectory* with the appropriate subdirectory for your PCB revision and FPGA type as detailed in the table above.

5.3 Communicating with the Host

There are three methods of communicating with the host.

5.3.1 Single Bit Communications with the Host

The RC1000-PP support software provides two macros to control single pins on the FPGA. The **PP1000ReadGPO()** macro expression reads the state of a pin on the FPGA and the **PP1000SetGPI()** macro procedure sets the state of a second FPGA pin. The following code fragment illustrates how to read the state of the GPO pin.

```
unsigned 1 Value;

Value = PP1000ReadGPO(); // 1 clock cycle
```

The following code fragment illustrates how to set the state of the GPI pin to 1.

```
PP1000SetGPI(1); // 1 clock cycle
```

Refer to section 4.5.1 for details of how to set and read the states of these pins from a host program.

5.3.2 Single Byte Communications with the Host

The RC1000-PP board has a single byte wide port in either direction between the host and FPGA. This port can be used to send short messages as control or status bytes between the two parties. The **PP1000WriteStatus()** function will send a byte from the FPGA to the host and the **PP1000ReadControl()** function will wait for a byte to be sent by the host.

Both macros are blocking and will only return when the operation has completed.

Refer to section 4.5.2 for details of how to send and receive single bytes in a Handel-C program on the FPGA.

Refer to the **addone** example program for code samples.

5.3.3 Accessing the External SRAM

Bulk data transfers between the host and the RC1000-PP SRAM can be performed by using the DMA support functions in the host support software. The Handel-C support software contains macros to allow access to the external SRAM.

Before accessing the external memory, ownership of the required memory bank must be requested and granted. The **PP1000RequestMemoryBank()** macro should be used to gain access to an external memory bank. The macro will only return when access to the bank has been granted. Access to the bank can be released by calling the **PP1000ReleaseMemoryBank()** macro.

The external memory can be accessed using the **PP1000ReadBank#()** and **PP1000WriteBank#()** macro procedures. For example, the following code fragment copies a word from address 0x1000 to address 0x2000 in bank 2:

```
unsigned 32 Data;

// Stage 1: request bank 2 (mask = 0b0100=0x4)
PP1000RequestMemoryBank(0x4);

// Stage 2: read word from address 0x1000
PP1000ReadBank2(Data, 0x1000);

// Stage 3: write word to address 0x2000
PP1000WriteBank2(0x2000, Data);

// Stage 4: release bank 2
PP1000ReleaseMemoryBank(0x4);
```

Refer to the Function Reference Manual for further details of the RC1000-PP Handel-C support software.

6. Utilities

6.1 Introduction

The RC1000-PP support software package contains a number of utility programs to aid with software development. The **gencfg** utility converts FPGA image files into a C header file to allow you to link your FPGA images into your application. The **loadfpga** utility will download an FPGA image to a card and set it running. This is useful if you wish to write an FPGA program without writing a host program to go with it. The **list** utility will list all the cards in a system. The **setid** utility will set the ID of a card to a new value. The **diag** utility can be used to perform diagnostics on a card.

This chapter details the use of each of these utilities.

6.2 The list Utility

The list utility simply lists all the cards present in the system detailing their card ID and serial number. Note that cards that are in use by another application may not have their serial number listed.

For example, typing list at a command prompt may return the following information:

```
Number of cards in system : 3

Card ID : 2        Serial Number : 0x00000003
Card ID : 43       Serial Number : 0x0000005a
Card ID : 99       Serial Number : 0x00000189
```

6.3 The setid Utility

The setid utility can be used to reprogram the ID of a card. Refer to section 4.2.1 for details of the use of card IDs. The general format of the command line is:

`setid` *SerialNumber CardID*

The serial number is required to uniquely identify the card to be reprogrammed since two cards can have the same card ID. For example, to reprogram the card with serial number 0x43 to have a card ID of 32, type the following:

```
setid 0x43 32
```

Under Windows 98, you will be asked to reboot the system upon completion of the setid utility to allow the changes to take effect. Rebooting is not necessary under Windows NT4.0.

6.4 The gencfg Utility

As described in chapter 4, the RC1000-PP support software provides functions to allow FPGA images to be included in a host executable file to avoid having many FPGA image files for one application. The **gencfg** utility takes an FPGA image file as its input and generates a C header file as its output. For example:

```
gencfg warp.bit warp.h
```

This will read the FPGA configuration file **warp.bit** and generate the C header file **warp.h**. The input file should be in Xilinx binary BIT file format (**.bit** extension).

The output file will have the following general format:

```
/*
 * Following lines generated by gencfg utility from warp.bit
 */
static unsigned long warpLength = 0x0003abf0;
static unsigned char warpBuffer[] = {
0xff, 0x20, 0x2b, 0x7d, 0x9f, 0x57, 0xfe, 0xfe,
0xbf, 0xaf, 0xeb, 0xfa, 0xfe, 0xbf, 0xaf, 0xeb,
0xfa, 0xfe, 0xbf, 0xb7, 0xf5, 0xfd, 0x7f, 0x5f,
........
```

These two definitions provide all the information required by the **PP1000RegisterImage()** function. For example, to include this image in your host program, you should:

1. Include the image definitions in your host program with the line:

   ```
   #include "warp.h"
   ```

2. Register the image with the host support software with the line:

   ```
   PP1000RegisterImage(warpBuffer,
                       warpLength,
                       &Image);
   ```

3. Configure the FPGA with the line:

   ```
   PP1000ConfigureFPGA(Handle, Image);
   ```

The **gencfg** utility will always name the two variables based on the destination filename. For example, the results of the command:

```
gencfg in.bit ..\..\dir1\fpga\dir2\out.h
```

would be:

```
/*
 * Following lines generated by gencfg utiltity from in.bit
 */
static unsigned long outLength = 0x0003abf0;
static unsigned char outBuffer[] = {
0xff, 0x20, 0x2b, 0x7d, 0x9f, 0x57, 0xfe, 0xfe,
0xbf, 0xaf, 0xeb, 0xfa, 0xfe, 0xbf, 0xaf, 0xeb,
0xfa, 0xfe, 0xbf, 0xb7, 0xf5, 0xfd, 0x7f, 0x5f,
........
```

The FPGA part type is automatically detected by the **PP1000RegisterImage()** function.

6.5 The loadfpga Utility

The **`loadfpga`** utility can be used to configure the RC1000-PP FPGA from the command prompt without having to write a host application. The general format of the command line is:

`loadfpga` {**`-i`** *CardID*} {**`-c`** *ClockRate*} *FileName*

Here, *CardID* is the ID of the card to configure. If this value is not set, the first free card will be configured.

ClockRate specifies the rate to set the programmable clock to before configuring the FPGA. This is optional for designs that do not use the programmable clock. Only MCLK may be programmed with this utility. Refer to the RC1000-PP Hardware Reference Manual for details of FPGA clock sources.

Filename specifies the FPGA image file to load. This file should be in Xilinx binary BIT format (**`.bit`** extension). Refer to the Xilinx FPGA software tools documentation for details of how to generate files of this type.

6.6 The diag Utility

The **diag** utility can be used to control an RC1000-PP from the command line. It allows board diagnostics to be performed without writing a host program. The general format of the command line is:

diag {*ScriptFile*}

When run without a script file name as a parameter, the utility will enter interactive mode and allows commands to be typed at a prompt. If a script file is specified then commands will be read from the script and executed.

6.6.1 Example Use of the diag Utility

To illustrate how the **diag** utility can replace a host program in some circumstances, consider the **addone** example program described in chapter 3. The following output shows the **addone** FPGA BIT file being used from the **diag** utility.

```
c:\rc1000pp\utils>diag

RC1000-PP Diagnostics Utility v1.00
(c) Celoxica Ltd 12 Jan 1999

> card 3

  Number of cards in system : 1

 * Card ID :     2  |  Serial Number : 0x00000003

> clock mclk 20000000

  Setting clock MCLK to 20000000Hz

> config addone.bit

  Configuring from file addone.bit

> wc 104

  Writing 104 to control port

> rs

  Status port returned 105

> quit

c:\rc1000pp\utils>
```

The following sections describe the commands that the **diag** utility understands.

6.6.2 The help Command

The **help** command lists all the commands that the **diag** utility understands. For example:

```
> help
RC1000-PP Diagnostics Utility v1.00
(c) Celoxica Ltd 12 Jan 1999

  bs                      Display bank ownership status
  card <SerialNum>        Set current card to serial number <SerialNum>
  clock <Source> <Rate>   Set clock rate. Source is 'mclk' or 'vclk'
  config <FileName>       Configure FPGA from file
  dm <Offset> {<Length>}  Display contents of memory
  em <Offset> <Value>     Edit memory word
  help                    Display help on commands (this screen)
  info                    Display card details
  list                    List all boards present
  quit                    Quit
  readgpi                 Read value from GPI pin
  rel <Mask>              Release memory banks in <Mask>
  req <Mask>              Request memory banks in <Mask>
  rs                      Read value from status port
  script <FileName>       Run commands from a script
  setgpo <Value>          Set value on GPO pin
  testdma                 Tests DMA transfers
  wc <Value>              Write value to control port
```

6.6.3 The quit Command

The **quit** command exits the **diag** utility.

6.6.4 The list Command

The **list** command lists all the RC1000-PP cards in the system. The currently selected card (see the card command) is marked with a '*'. The **list** command displays the following information:

```
> list

  Number of cards in system : 1

   Card ID :     2  |  Serial Number : 0x00000003
```

6.6.5 The card Command

The **card** command opens one of the cards in the system to be the target of all future operations. The card is identified by its serial number which is factory set and guaranteed to be unique across all boards. The currently selected card is marked with a '*'. The **card** command displays the following information:

```
> card 3

  Number of cards in system : 1

 * Card ID :     2  |  Serial Number : 0x00000003
```

6.6.6 The info Command

The **info** command lists the hardware details of a card. The card command can be used to select the target of this operation. For example:

```
> card 3

  Number of cards in system : 1

 * Card ID :     2 |  Serial Number : 0x00000003

> info
        CardID          : 2
        PMC1Fitted      : 0
        PMC2Fitted      : 0
        FPGAType        : 1
        Physical base   : 0xe9000000
        Board Rev       : 2.0
        Logic Rev       : 0.1
        Serial Number   : 3
        Bank 0 size     : 0x200000
        Bank 1 size     : 0x200000
        Bank 2 size     : 0x200000
        Bank 3 size     : 0x200000
```

6.6.7 The config Command

The **config** command configures the FPGA on the target board from a file. The target board can be set with the **card** command. The file must be in Xilinx binary BIT format (**.bit** extension).

For example:

```
> card 3

  Number of cards in system : 1

 * Card ID :     2  |  Serial Number : 0x00000003

> config addone.bit

  Configuring from file addone.bit
```

6.6.8 The clock Command

The **clock** command sets one of the programmable clock sources on the board to a given rate. The target board can be set with the **card** command. The clock source can be either '**mclk**' or '**vclk**'. The frequency can be from 400,000 to 100,000,000 Hertz. Refer to the **PP1000SetClockRate()** function in the Function Reference Manual for details of the different clock sources.

For example:

```
> card 3

  Number of cards in system : 1

 * Card ID :     2  |  Serial Number : 0x00000003

> clock mclk 20000000

  Setting clock MCLK to 20000000Hz
```

6.6.9 The setgpo Command

The **setgpo** command sets the state of the GPO pin on the FPGA on the target board. The target board can be set with the **card** command. Refer to the **PP1000SetGPO()** function in the Function Reference Manual for details of the GPO or USERO pin. For example:

```
> setgpo 0

  Setting GPO to 0
```

6.6.10 The readgpi Command

The **readgpi** command reads the state of the GPI or USERI pin on the FPGA on the target board. The target board can be set with the **card** command. Refer to the **PP1000ReadGPI()** function in the Function Reference Manual for details of the GPI pin. For example:

```
> readgpi

  GPI pin is 1
```

6.6.11 The wc Command

The **wc** command writes a byte to the control port of a board. The parameter is the byte to write. The target board can be set with the **card** command. The **wc** command will wait until the FPGA has read the byte before returning. For example:

```
> wc 23

  Writing 23 to control port
```

6.6.12 The rs Command

The **rs** command reads a byte from the status port of a board. The target board can be set with the **card** command. The **rs** command will wait until the FPGA has written to the status port before returning. For example:

```
> rs

  Status port returned 24
```

6.6.13 The req Command

The **req** command requests ownership of a set of memory banks. The parameter is a 4 bit mask of the banks to request. Bit 0 (the LSB) should be set to 1 to request bank 0, bit 1 should be set to 1 to request bank 1 and so on. The target board can be set with the **card** command. The command will wait until the requested banks are owned by the host before returning.

For example.

```
> req 3

  Memory Bank Status

          Bank 3   Bank 2  Bank 1  Bank 0
    Host                      *       *
    FPGA
```

6.6.14 The rel Command

The **rel** command releases ownership of a set of memory banks. The parameter is a 4 bit mask of the banks to release. Bit 0 (the LSB) should be set to 1 to release bank 0, bit 1 should be set to 1 to release bank 1 and so on. The target board can be set with the **card** command.

For example.

```
> rel 3

  Memory Bank Status

         Bank 3   Bank 2  Bank 1  Bank 0
  Host
  FPGA
```

6.6.15 The bs Command

The **bs** command returns details of the ownership status of the memory banks on a card. The target board can be set with the **card** command. For example:

```
> bs

  Memory Bank Status

         Bank 3   Bank 2  Bank 1  Bank 0
  Host             *               *
  FPGA
```

In this example, the host owns banks 0 and 2 and the FPGA owns none of the banks.

6.6.16 The dm Command

The **dm** command displays the contents of a range of addresses in the SRAM of a board. The first parameter gives a byte offset into the SRAM for the start of the display and the second parameter specifies the number of bytes to display. If the second parameter is omitted then 256 bytes will be displayed. The target board can be set with the **card** command.

The **dm** command will automatically request ownership of all the banks that the address range covers.

For example:

```
> dm 8 64

  0x00000000  :  ********  ********  00000002  00000003
  0x00000010  :  00000004  00000005  00000006  00000007
  0x00000020  :  00000008  00000009  0000000a  0000000b
  0x00000030  :  0000000c  0000000d  0000000e  0000000f
  0x00000040  :  00000010  00000011  ********  ********
```

6.6.17 The em Command

The **em** command sets the value of a 32 bit word in the SRAM of a board. The first parameter gives a byte offset into the SRAM for the word to edit and the second parameter gives the value to set the word to. The target board can be set with the **card** command.

The **em** command will automatically request ownership of the bank that contains the address.

For example:

```
> dm 8 64

  0x00000000  :  ********  ********  00000002  00000003
  0x00000010  :  00000004  00000005  00000006  00000007
  0x00000020  :  00000008  00000009  0000000a  0000000b
  0x00000030  :  0000000c  0000000d  0000000e  0000000f
  0x00000040  :  00000010  00000011  ********  ********

> em 0x24 0x12345678

  Writing 0x12345678 to offset 0x00000024

> dm 8 64

  0x00000000  :  ********  ********  00000002  00000003
  0x00000010  :  00000004  00000005  00000006  00000007
  0x00000020  :  00000008  12345678  0000000a  0000000b
  0x00000030  :  0000000c  0000000d  0000000e  0000000f
  0x00000040  :  00000010  00000011  ********  ********
```

6.6.18 The testdma Command

The **testdma** command performs a memory test on all the fitted memory banks on a board. The target board can be set with the **card** command.

For example:

```
> testdma

  Testing bank 0 (size : 0x200000)
    Transfer rate to card = 102.41Mb/s
    Transfer rate from card = 109.41Mb/s

  Testing bank 1 (size : 0x200000)
    Transfer rate to card = 103.20Mb/s
    Transfer rate from card = 110.31Mb/s
```

6.6.19 The script Command

The **script** command runs commands from a file specified as the parameter. Commands must appear one per line in the script file exactly as they would if they were typed at the command prompt. A line in the script may be designated as a comment by using a # character as the first character in the line. For example, the following script:

```
# Select card
card 3
# Request some memory banks
req 0x3
req 0x8
# Release some memory banks
rel 0xa
```

results in the following output:

```
> script test

> card 3

  Number of cards in system : 1

 * Card ID :     2  |  Serial Number : 0x00000003

> req 0x3

  Memory Bank Status

          Bank 3   Bank 2   Bank 1   Bank 0
    Host                     *        *
    FPGA

> req 0x8

  Memory Bank Status

          Bank 3   Bank 2   Bank 1   Bank 0
    Host    *                *        *
    FPGA

> rel 0xa

  Memory Bank Status

          Bank 3   Bank 2   Bank 1   Bank 0
    Host                              *
    FPGA
```